CHAPTER 1

Bootstrap Scaffolding

What Is Bootstrap?

Bootstrap is an open source product from Mark Otto and Jacob Thornton who, when it was initially released, were both employees at Twitter. There was a need to standardize the frontend toolsets of engineers across the company. In the launch blog post, Mark Otto introduced the project like this:

> In the earlier days of Twitter, engineers used almost any library they were familiar with to meet front-end requirements. Inconsistencies among the individual applications made it difficult to scale and maintain them. Bootstrap began as an answer to these challenges and quickly accelerated during Twitter's first Hackweek. By the end of Hackweek, we had reached a stable version that engineers could use across the company.
>
> — Mark Otto
> *https://dev.twitter.com/*

Since Bootstrap launched in August 2011, it has taken off in popularity. It has evolved from being an entirely CSS-driven project to include a host of JavaScript plugins and icons that go hand in hand with forms and buttons. At its base, it allows for responsive web design and features a robust 12-column, 940px-wide grid. One of the highlights is the build tool on Bootstrap's website, where you can customize the build to suit your needs, choosing which CSS and JavaScript features you want to include on your site. All of this allows frontend web development to be catapulted forward, building on a stable foundation of forward-looking design and development. Getting started with Bootstrap is as simple as dropping some CSS and JavaScript into the root of your site.

For someone starting a new project, Bootstrap comes with a handful of useful elements. Normally, when I start a project, I start with tools like Eric Meyer's Reset CSS and get going on my web project. With Bootstrap, you just need to include the *bootstrap.css* CSS file and, optionally, the *bootstrap.js* JavaScript file into your website and you are ready to go.

Bootstrap File Structure

```
bootstrap/
       ├── css/
       │   ├── bootstrap.css
       │   ├── bootstrap.min.css
       ├── js/
       │   ├── bootstrap.js
       │   ├── bootstrap.min.js
       ├── img/
       │   ├── glyphicons-halflings.png
       │   ├── glyphicons-halflings-white.png
       └── README.md
```

The Bootstrap download includes three folders: css, js, and img. For simplicity, add these to the root of your project. Minified versions of the CSS and JavaScript are also included. It is not necessary to include both the uncompressed and the minified versions. For the sake of brevity, I use the uncompressed version during development and then switch to the compressed version in production.

Basic HTML Template

Normally, a web project looks something like this:

```
<!DOCTYPE html>
<html>
        <head>
                <title>Bootstrap 101 Template</title>
        </head>
        <body>
                <h1>Hello, world!</h1>
        </body>
</html>
```

With Bootstrap, we include the link to the CSS stylesheet and the JavaScript:

```
<!DOCTYPE html>
<html>
        <head>
                <title>Bootstrap 101 Template</title>
                <link href="css/bootstrap.min.css" rel="stylesheet">
        </head>
        <body>
                <h1>Hello, world!</h1>
                <script src="js/bootstrap.min.js"></script>
        </body>
</html>
```

Don't forget the HTML5 Doctype.

By including <!DOCTYPE html>, all modern browsers are put into standards mode.

Global Styles

With Bootstrap, a number of items come prebuilt. Instead of using the old reset block that was part of the Bootstrap 1.0 tree, Bootstrap 2.0 uses Normalize.css, a project from Nicolas Gallagher that is part of the HTML5 Boilerplate. This is included in the *bootstrap.css* file.

In particular, the following default styles give special treatment to typography and links:

- `margin` has been removed from the body, and content will snug up to the edges of the browser window.
- `background-color: white;` is applied to the body.
- Bootstrap is using the `@baseFontFamily`, `@baseFontSize`, and `@baseLineHeight` attributes as our typographic base. This allows the height of headings and other content around the site to maintain a similar line height.
- Bootstrap sets the global link color via `@linkColor` and applies link underlines only on `:hover`.

Remember, if you don't like the colors or want to change a default, this can be done by changing the globals in any of the *.less* files. To do this, update the *scaffolding.less* file or overwrite colors in your own stylesheet.

Default Grid System

The default Bootstrap grid (see Figure 1-1) system utilizes 12 columns, making for a 940px-wide container without responsive features enabled. With the responsive CSS file added, the grid adapts to be 724px or 1170px wide, depending on your viewport. Below 767px viewports, such as the ones on tablets and smaller devices, the columns become fluid and stack vertically. At the default width, each column is 60 pixels wide and offset 20 pixels to the left. An example of the 12 possible columns is in Figure 1-1.

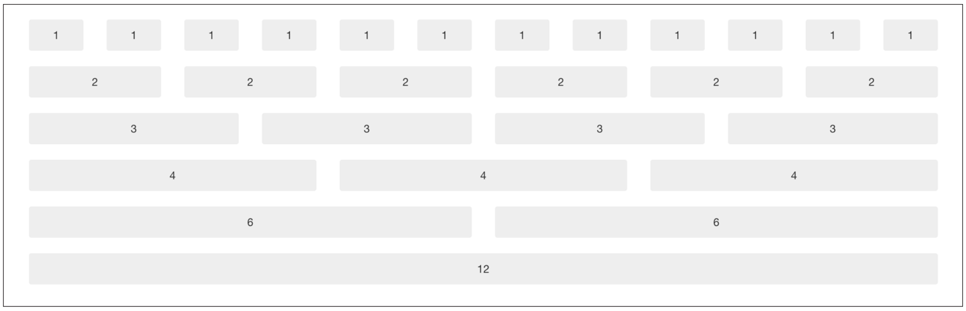

Figure 1-1. Default grid

Basic Grid HTML

To create a simple layout, create a container with a `<div>` that has a class of `.row` and add the appropriate amount of `.span*` columns. Since we have a 12-column grid, we just need the amount of `.span*` columns to equal 12. We could use a 3-6-3 layout, 4-8, 3-5-4, 2-8-2… we could go on and on, but I think you get the gist.

The following code shows `.span8` and `.span4`, which adds up to 12:

```
<div class="row">
  <div class="span8">...</div>
  <div class="span4">...</div>
</div>
```

Offsetting Columns

You can move columns to the right using the `.offset*` class. Each class moves the span over that width. So an `.offset2` would move a `.span7` over two columns (see Figure 1-2):

```
<div class="row">
  <div class="span2">...</div>
  <div class="span7 offset2">...</div>
</div>
```

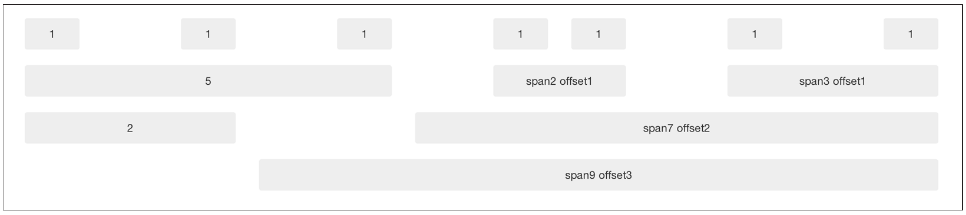

Figure 1-2. Offset grid

Nesting Columns

To nest your content with the default grid, inside of a `.span*`, simply add a new `.row` with enough `.span*` that it equals the number of spans of the parent container (see Figure 1-3):

```
<div class="row">
  <div class="span9">
    Level 1 of column
    <div class="row">
      <div class="span6">Level 2</div>
      <div class="span3">Level 2</div>
    </div>
  </div>
</div>
```

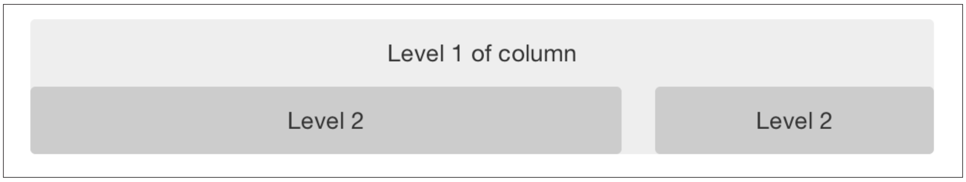

Figure 1-3. Nesting grid

Fluid Grid System

The fluid grid system uses percentages instead of pixels for column widths. It has the same responsive capabilities as our fixed grid system, ensuring proper proportions for key screen resolutions and devices. You can make any row "fluid" by changing `.row` to `.row-fluid`. The column classes stay exactly the same, making it easy to flip between fixed and fluid grids. To offset, you operate in the same way as the fixed grid system—add `.offset*` to any column to shift by your desired number of columns:

```
<div class="row-fluid">
  <div class="span4">...</div>
  <div class="span8">...</div>
</div>

<div class="row-fluid">
  <div class="span4">...</div>
  <div class="span4 offset2">...</div>
</div>
```

Nesting a fluid grid is a little different. Since we are using percentages, each `.row` resets the column count to 12. For example, if you were inside a `.span8`, instead of two `.span4` elements to divide the content in half, you would use two `.span6` divs (see

Figure 1-4). This is the case for responsive content, as we want the content to fill 100% of the container:

```
<div class="row-fluid">
  <div class="span8">
              <div class="row">
                      <div class="span6">...</div>
                      <div class="span6">...</div>
              </div>
  </div>
</div>
```

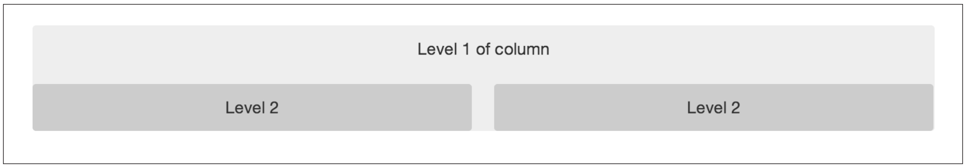

Figure 1-4. Nesting fluid grid

Container Layouts

To add a fixed-width, centered layout to your page, simply wrap the content in `<div class="container">...</div>`. If you would like to use a fluid layout but want to wrap everything in a container, use the following: `<div class="container-fluid">...</div>`. Using a fluid layout is great when you are building applications, administration screens, and other related projects.

Responsive Design

To turn on the responsive features of Bootstrap, you need to add a `<meta>` tag to the `<head>` of your web page. If you haven't downloaded the compiled source, you will also need to add the responsive CSS file. An example of required files looks like this:

```
<!DOCTYPE html>
<html>
        <head>
                <title>My amazing Bootstrap site!</title>
                <meta name="viewport" content="width=device-width,
              initial-scale=1.0">
                <link href="/css/bootstrap.css" rel="stylesheet">
                <link href="/css/bootstrap-responsive.css" rel="stylesheet">
        </head>
```

If you get started and find that the Bootstrap responsive features aren't working, make sure that you have these tags. The responsive features aren't added by default at this time because not everything needs to be responsive. Instead of encouraging developers to remove this feature, the authors of Bootstrap decided that it was best to enable it as needed.

What Is Responsive Design?

Responsive design is a method for taking all of the existing content that is on the page and optimizing it for the device that is viewing it. For example, the desktop not only gets the normal version of the website, but it might also get a widescreen layout, optimized for the larger displays that many people have attached to their computers. Tablets get an optimized layout, taking advantage of their portrait or landscape layouts. And then with phones, you can target their much narrower width. To target these different widths, Bootstrap uses CSS media queries to measure the width of the browser viewport and then, using conditionals, changes which parts of the stylesheets are loaded. Using the width of the browser viewport, Bootstrap can then optimize the content using a combination of ratios or widths, but it mostly relies on *min-width* and *max-width* properties.

At the core, Bootstrap supports five different layouts, each relying on CSS media queries. The largest layout has columns that are 70 pixels wide, contrasting with the 60 pixels of the normal layout. The tablet layout brings the columns to 42 pixels wide, and when narrower than that, each column goes fluid, meaning the columns are stacked vertically and each column is the full width of the device (see Table 1-1).

Table 1-1. Responsive media queries

Label	Layout width	Column width	Gutter width
Large display	1200px and up	70px	30px
Default	980px and up	60px	20px
Portrait tablets	768px and up	42px	20px
Phones to tablets	767px and below	Fluid columns, no fixed widths	
Phones	480px and below	Fluid columns, no fixed widths	

To add custom CSS based on the media query, you can either include all rules in one CSS file via the media queries below, or use entirely different CSS files:

```css
/* Large desktop */
@media (min-width: 1200px) { ... }

/* Portrait tablet to landscape and desktop */
@media (min-width: 768px) and (max-width: 979px) { ... }

/* Landscape phone to portrait tablet */
@media (max-width: 767px) { ... }
```

```
/* Landscape phones and down */
@media (max-width: 480px) { ... }
```

For a larger site, you might want to divide each media query into a seperate CSS file. In the HTML file, you can call them with the `<link>` tag in the head of your document. This is useful for keeping file sizes smaller, but it does potentially increase the HTTP requests if the site is responsive. If you are using LESS to compile the CSS, you can have them all processed into one file:

```
<link rel="stylesheet" href="base.css" />
<link rel="stylesheet" media="(min-width: 1200px)" href="large.css" />
<link rel="stylesheet" media="(min-width: 768px) and (max-width: 979px)"
      href="tablet.css" />
<link rel="stylesheet" media="(max-width: 767px)" href="tablet.css" />
<link rel="stylesheet" media="(max-width: 480px)" href="phone.css" />
```

Helper classes

Bootstrap also includes a handful of helper classes for doing responsive development (see Table 1-2). Use these sparingly. A couple of use cases that I have seen involve loading custom elements based on certain layouts. Perhaps you have a really nice header on the main layout, but on mobile you want to pare it down, leaving only a few of the elements. In this scenario, you could use the `.hidden-phone` class to hide either parts or entire dom elements from the header.

Table 1-2. Media queries helper classes

Class	Phones	Tablets	Desktops
.visible-phone	Visible	Hidden	Hidden
.visible-tablet	Hidden	Visible	Hidden
.visible-desktop	Hidden	Hidden	Visible
.hidden-phone	Hidden	Visible	Visible
.hidden-tablet	Visible	Hidden	Visible
.hidden-desktop	Visible	Visible	Hidden

There are two major ways that you could look at doing development. The mantra that a lot of people are shouting now is that you should start with mobile, build to that platform, and let the desktop follow. Bootstrap almost forces the opposite, where you would create a full-featured desktop site that "just works."

If you are looking for a strictly mobile framework, Bootstrap is still a great resource.